CS 575/475

Chih Hsuan Huang

huanchih@oregonstate.edu

ID: 934554197

Project #4

Vectorized Array Multiplication and Multiplication/Reduction using SSE

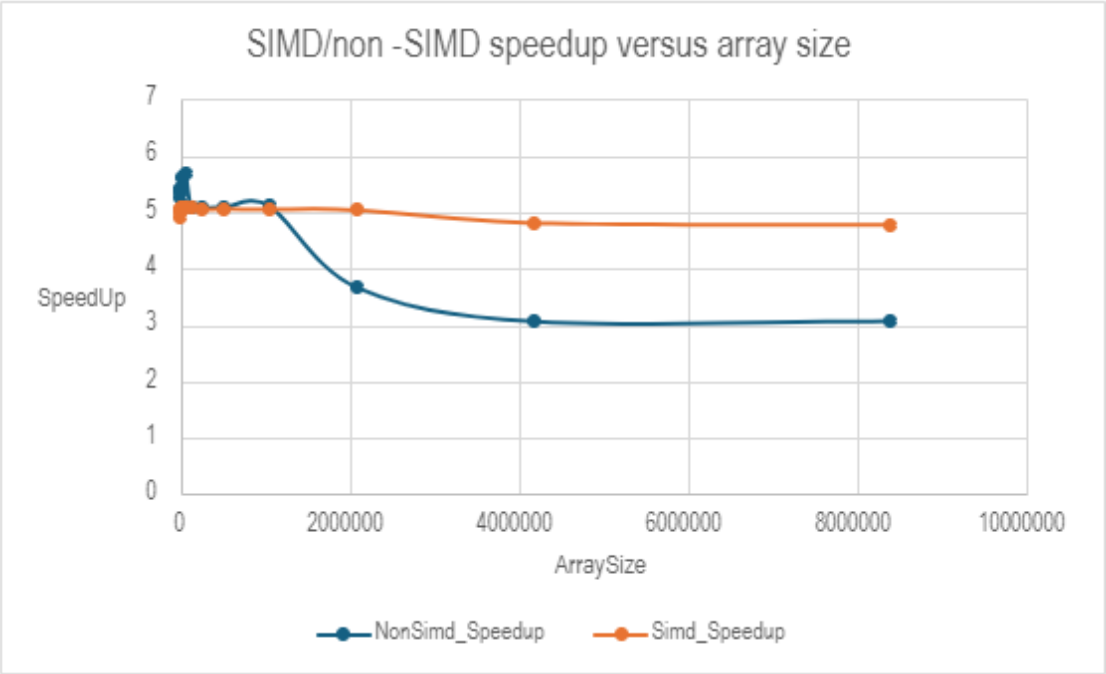
1. **What machine you ran this on**

   rabbit

2. **Show the 2 tables of performances for each array size and the corresponding speedups**

ArraySize	NonSimd_megaMults	Simd_megaMults	NonSimd_Speedup	NonSimd_megaMultAdds	Simd_megaMultAdds	Simd_Speedup
1024	196.39	1030.17	5.25	208.98	1023.99	4.9
2048	134.69	723.41	5.37	143.57	721.12	5.02
4096	142.02	769.06	5.41	154.93	785.88	5.07
8192	175.29	950.24	5.42	221.44	1121.58	5.06
16384	133.61	727.05	5.44	155.06	788.26	5.08
32768	216.03	1216.51	5.63	254.8	1296.31	5.09
65536	174.16	991.38	5.69	232.61	1184.26	5.09
131072	328.16	1672.67	5.1	354.5	1805.97	5.09
262144	328.29	1668.99	5.08	354.5	1788.51	5.05
524288	327.81	1669.16	5.09	354.03	1791.73	5.06
1048576	327.39	1676.88	5.12	353.87	1786.6	5.05
2097152	322.88	1181.46	3.66	353.53	1780.17	5.04
4194304	317.04	969.46	3.06	348.51	1677.63	4.81
8388608	315.57	969.38	3.07	346.51	1655.68	4.78

3. **Show the graphs (or graph) of SIMD/non-SIMD speedup versus array size (either one graph with two curves, or two graphs each with one curve)**



4. **What patterns are you seeing in the speedups?**

At smaller array sizes, the speedup values for SIMD and non-SIMD are higher, When processing data smaller than 1048576, the acceleration of SIMD and non-SIMD still increases slightly and as the array size increases, the speedup values decrease. This pattern is particularly evident in non-SIMD speedups, with high speedups starting with small arrays gradually declining significantly. Compared to non-SIMD, SIMD acceleration shows greater consistency across different array sizes. After an initial dip, SIMD speedup stabilizes between approximately 5.00 and 5.10 on larger arrays. In contrast, the non-SIMD speedup drops significantly, with the sudden drop in speed to 3.66 at array size 2097152.

5. **Are they consistent across a variety of array sizes?**

SIMD and non-SIMD acceleration are not completely consistent across array sizes. Although the efficiency of both SIMD and non-SIMD operations decreases as the array size increases, this decrease is more significant in non-SIMD operations, indicating that it is more affected by

increased data processing and memory access times than SIMD operations. The impact can be speculated to be an increase in processor latency while waiting for data to be transferred from memory due to increased task complexity.

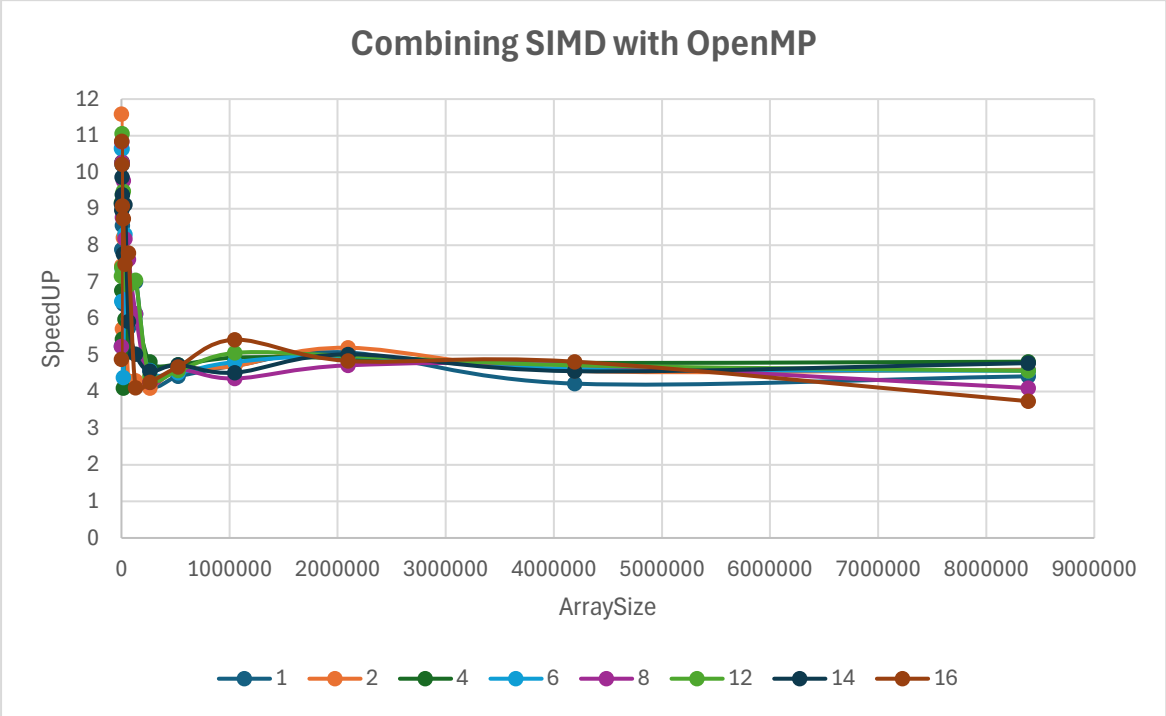
6. **Why or why not, do you think?**

SIMD is almost always approaching a 5 or higher speedup when processing arrays below 1048576. However, after 2097156, the acceleration value begins to decrease significantly. This decrease in efficiency may be due to the fact that larger arrays require more frequent memory accesses, and the increase in memory access time causes the operation to slow down. This may be because as array sizes increase, the overhead of managing larger data volumes may affect memory access patterns, potentially leading to bandwidth saturation or increased latency, which particularly affects non-SIMD operations. Smaller arrays may fit better in CPU cache, making data access and processing faster. As the array size increases, the likelihood of cache misses increases, reducing the speedup effect. SIMD operations can process multiple data points in parallel using a single instruction and are very efficient for operations that are easily parallelizable and have well-aligned data. This explains why SIMD maintains relatively stable speedups even as array sizes increase, although still showing some degradation at very large scales, possibly due to similar memory and cache issues

Extra credit:

ArraySize	1	2	4	6	8	12	14	16
1024	5.25	11.59	9.17	10.66	5.24	7.16	9.13	4.89
2048	7.89	7.44	6.77	6.47	10.83	7.4	8.97	10.85
4096	10.21	10.27	10.28	10.64	10.27	11.06	9.87	10.22
8192	8.55	5.72	5.44	7.31	8.77	9.4	9.39	9.08
16384	6.4	8.21	4.1	4.39	9.77	9.48	7.77	8.73
32768	5.31	6.61	5.98	8.3	8.18	7.72	9.12	7.49
65536	5.75	4.29	5.09	5.98	7.62	6.91	5.92	7.79
131072	7.01	4.3	6.12	6.13	6.13	7.05	5.03	4.11

262144	4.27	4.1	4.82	4.58	4.26	4.42	4.56	4.26
524288	4.42	4.58	4.74	4.52	4.61	4.58	4.73	4.68
1048576	4.71	4.7	4.93	4.81	4.36	5.06	4.52	5.42
2097152	5.07	5.2	4.92	4.97	4.72	4.98	5.02	4.84
4194304	4.22	4.57	4.79	4.62	4.73	4.72	4.56	4.82
8388608	4.42	4.6	4.82	4.57	4.1	4.57	4.78	3.74



In the table, the array sizes range from 1024 to 8388608, listing the speedup factor from 1 thread to 16 threads. In small arrays, using more threads can significantly increase the calculation speed, for example, when using 16 threads, the acceleration factor reaches 10.85 times. This shows that for smaller data sets, increasing the number of threads can significantly improve performance. As the array size increases, the speedup factor for all thread configurations levels off, remaining between 4 and 6x in most cases. This may be because as the amount of data increases, the overhead of memory access and data management gradually becomes a performance bottleneck.